DEPARTMENT OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA

CONTROL METHODOLOGIES FOR LARGE SPACE STRUCTURES

By

Griffith J. McRee, Principal Investigator

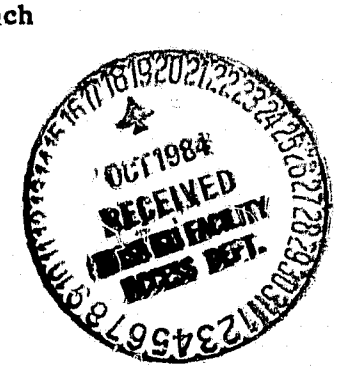
and

Edmond Altonji

Final Report
For the period ended September 1983

Prepared for the
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NAG-1-318
Dr. Raymond C. Montgomery, Technical Monitor
FDCD - Spacecraft Controls Branch



August 1984

DEPARTMENT OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA

CONTROL METHODOLOGIES FOR LARGE SPACE STRUCTURES

By

Griffith J. McRee, Principal Investigator

and

Edmond Altonji

Final Report
For the period ended September 1983

Prepared for the
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NAG-1-318
Dr. Raymond C. Montgomery, Technical Monitor
FDCD - Spacecraft Controls Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508

August 1984

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

This report describes work done at Old Dominion University under Contract NAS1-318 to the NASA-Langley Research Center. The work should be considered as preliminary to the larger tasks of integrating the controlled motor system described herein into the system which will dampen oscillations of a large free plate in space. The authors are responsible for the recording of the information in this report; however, several others should be credited with significant contributions to the work. First, our thanks go to Dr. Ray Montgomery of the Langley Research Center who provided us with the development system as well as invaluable advice on its use. Two other graduate students at Old Dominion University played significant roles in this work. Sophie Capelle did much of the initial conceptual work and Mark Motter later became a significant factor in the digital system implementation. Our thanks also go to these two without whom this work would not have been accomplished.

1. INTRODUCTION

The objectives of this research were to develop techniques of controlling a dc-motor driven flywheel which would apply torque to the structure to which it was mounted. The motor control system was to be implemented using a microprocessor based controller. The purpose of the torque applied by this system was to dampen oscillations of the structure to which it was mounted.

Before the work was terminated due to the unavailability of equipment, a system was developed and partially tested which would provide tight control of the flywheel velocity when it received a velocity command in the form of a voltage.

The procedure followed in this development was to first model the motor and flywheel system on an analog computer located at Old Dominion University. Then, prior to the time the microprocessor development system was available, an analog control loop was designed and tested. When the microprocessor development system became available, the analog control loop was replaced by the microprocessor and the system was partially tested.

2. ANALOG SIMULATION

2.1 Motor Model

From tests and specifications on the motor, the following transfer function for the motor was determined. (See Fig. 2)

$$\frac{K}{JS + B}$$

where

$$K = 4.02 \text{ oz in/A}$$
$$J = 5.096 \times 10^{-2} \text{ oz in s}^2$$
$$B = 2.0 \times 10^{-3} \text{ oz in s}$$

A current amplifier was employed to power the motor. Its gain, G, was 0.5.

Thus

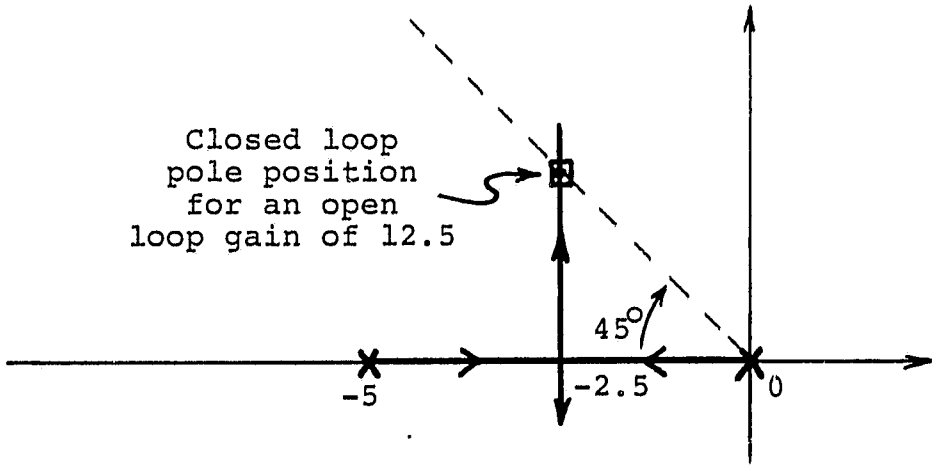
$$\frac{\omega}{V_{in}} = \frac{GK}{Js + B} = \frac{40}{s + 0.04}$$

2.2 Compensator

It is desirable to move the pole further from the imaginary axis to speed up the transient response time. To accomplish this output feedback and a lead compensator is used.

Compensator: $C(s) = \frac{A(s + 0.04)}{s(s + 5)}$

This provides a settling time of 1.6 seconds (to within 2% of steady state value). Choosing a damping coefficient of $\zeta = 0.707$, we obtain the root locus in Fig. 1.



For an open loop gain of 12.5,

$$40A = 12.5,\ A = 0.312$$

thus,

$$C(s) = \frac{0.312(s + 0.04)}{s(s+5)}$$

Fig. 3 shows the actual motor model with the compensator. Fig. 4 describes the response time of the motor to a step input without the compensator. Fig. 5 describes the response time with the compensator.

BLOCK DIAGRAM OF CLOSED-LOOP SYSTEM

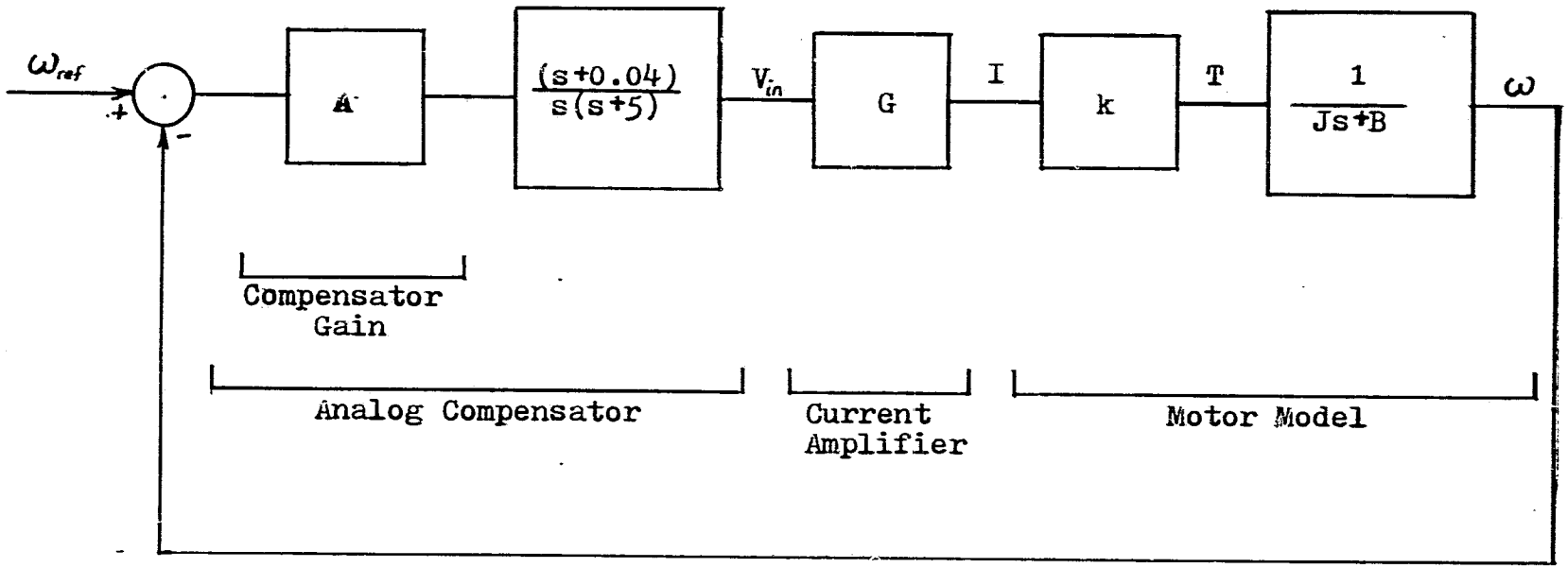


Fig. 2

Simulation Diagram for Closed-Loop System With Lead Compensation

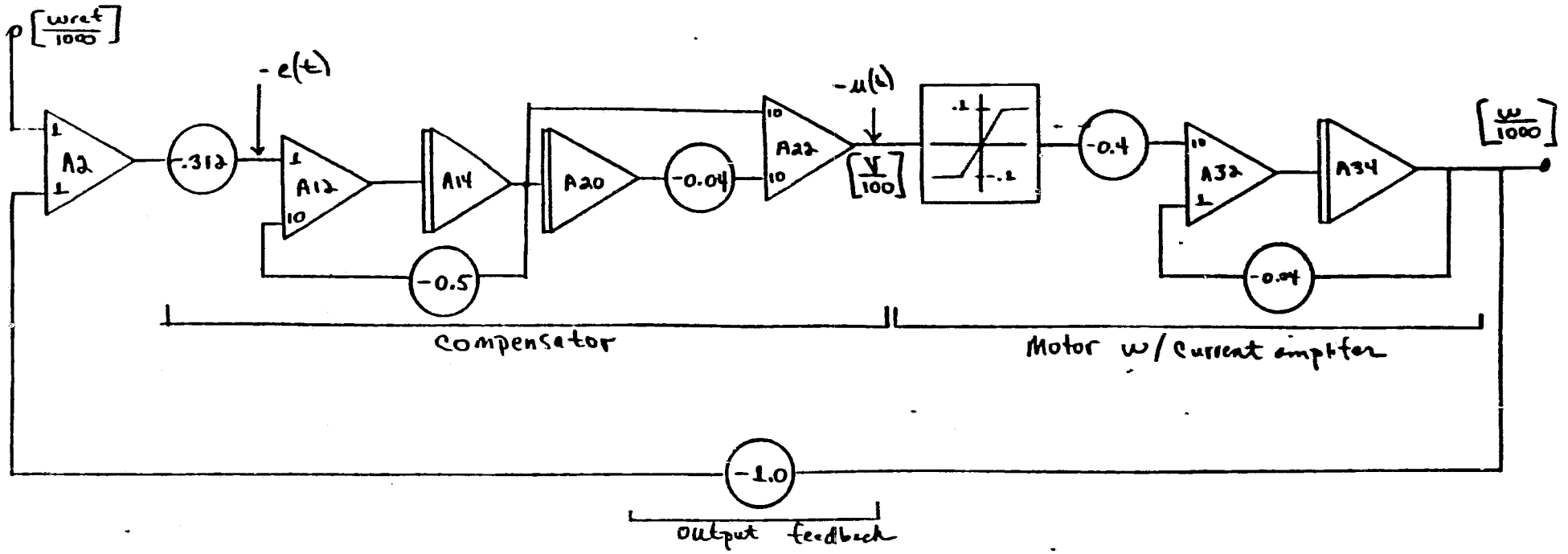


Fig. 3

MOTOR SPEED vs TIME

Conditions:
1. Motor driven by current amplifier only.
2. Step input with flywheel.

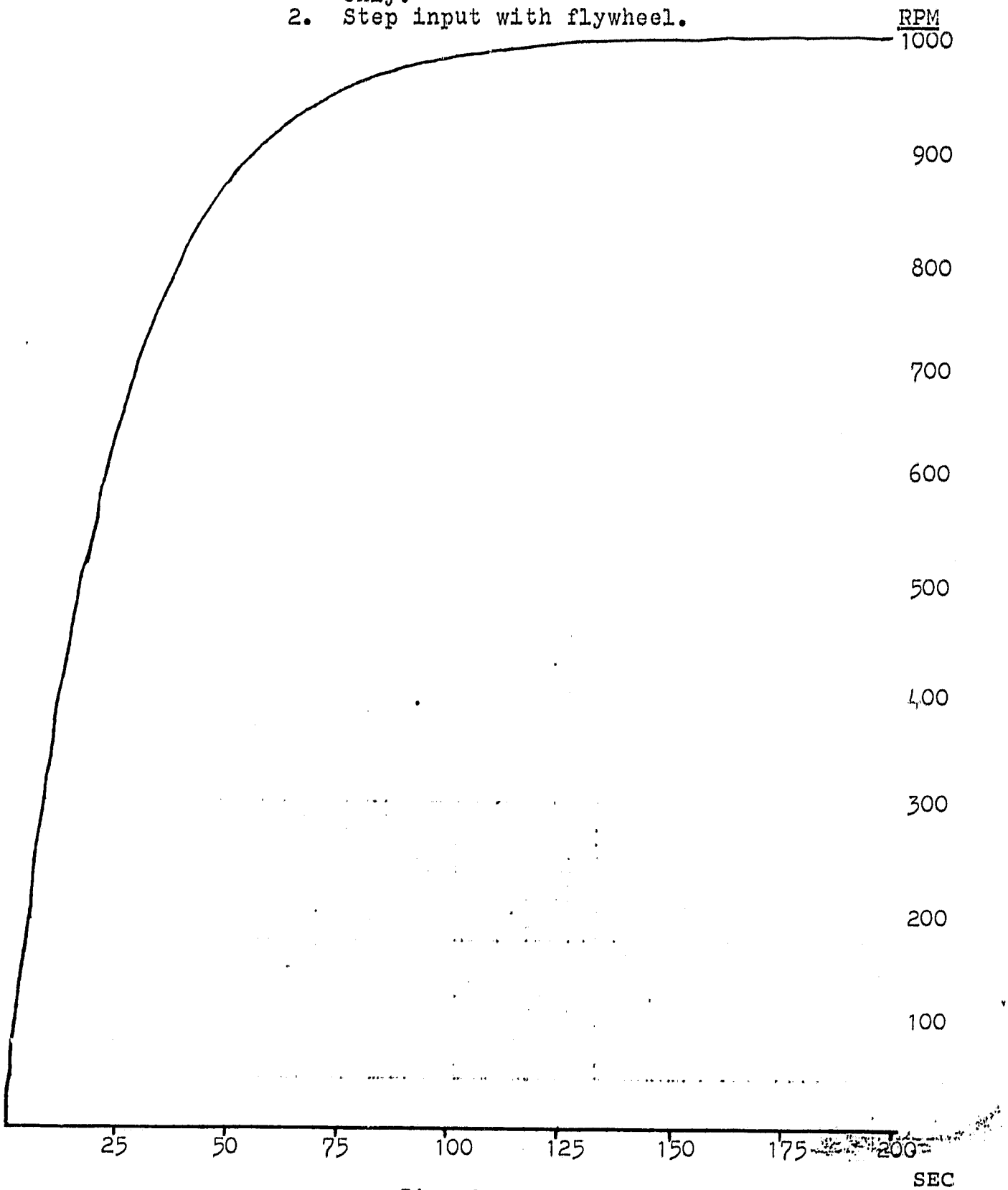


Fig. 4

MOTOR SPEED vs TIME (1)
INPUT VOLTAGE vs TIME (2)

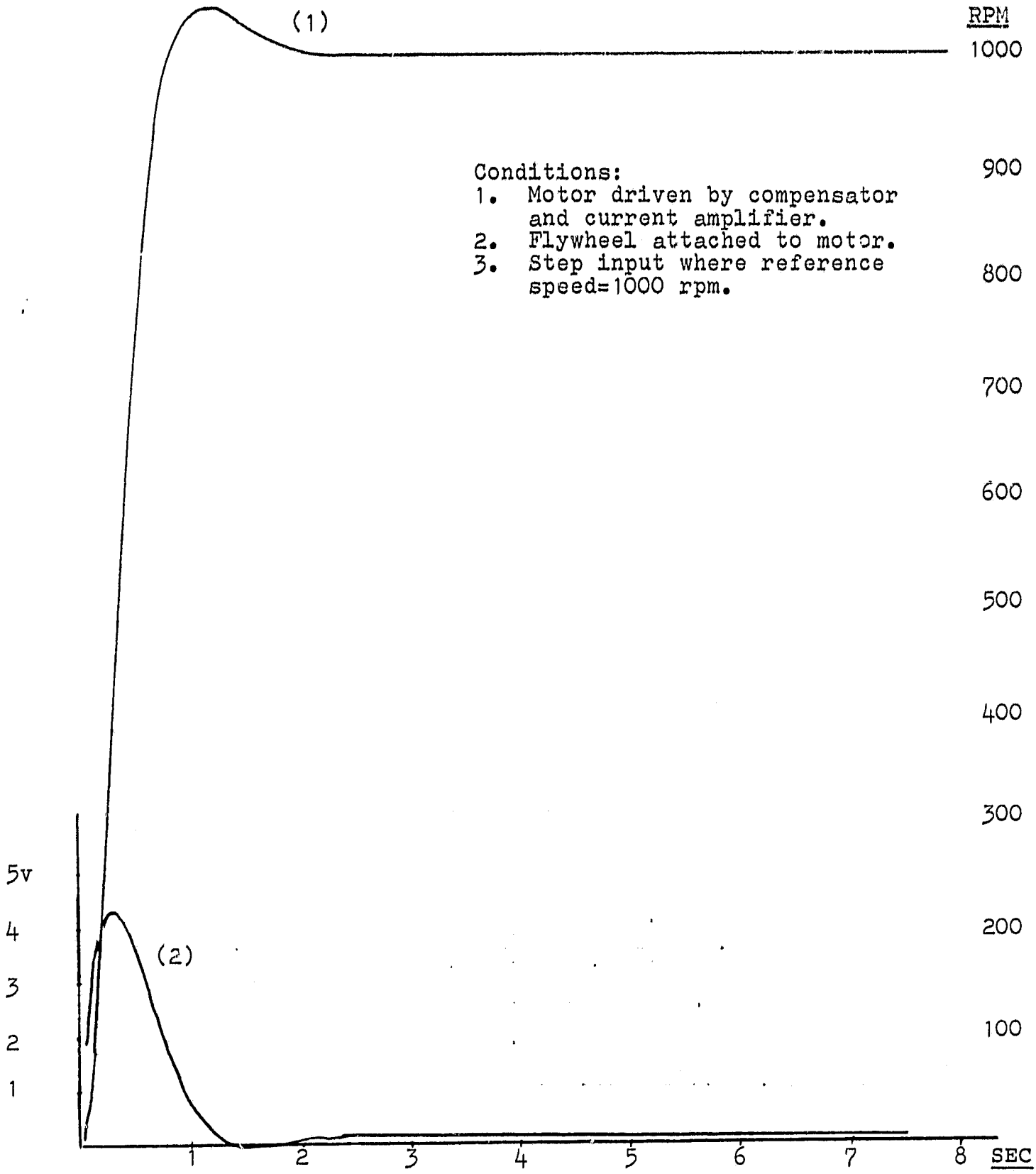


Fig. 5

3. DIGITAL DEVELOPMENT SYSTEM

3.1 System Description

The M68KDM design module was used to develop the digital control loop. On-board components include the following:

One Motorola M68000 16-bit microprocessor
32-kbytes of RAM
One 8-kbyte System Monitor (MACSBUG)
Two Synchronous Communication Terminals (ACIA)
Two 16-bit Peripheral Interface Devices (PID)
One Programmable Timer Module (PTM) with three timers (T1,T2,T3)

The system includes an assortment of pin-outs for control signals, voltages and ground terminals. The system clock and baud rate can be changed to suit user requirements as well.

3.2 System Connect

The NASA-LRC computer was connected through the "Host" ACIA, and our terminal was connected through the "Terminal" ACIA. With this arrangement, only one command was necessary to switch between the NASA computer and the M68KDM. This made programming easier since it was possible to jump quickly from the assembler to the microprocessor to check programs. The NASA-LRC made available compilers that handle both Pascal and MC68000 assembler code. It is even possible to use both in the same program by linking them in the downloading process. For our purposes the assembler code was sufficient, though we did experiment with the Pascal and found it very useful for more complicated programming.

3.3 MC68000 Microprocessor Description

A more detailed explanation of the microprocessor itself will prove helpful in understanding the programming. The following facilities are available to the

Registers

(8) 32-bit Data Registers

(7) 32-bit Address Registers

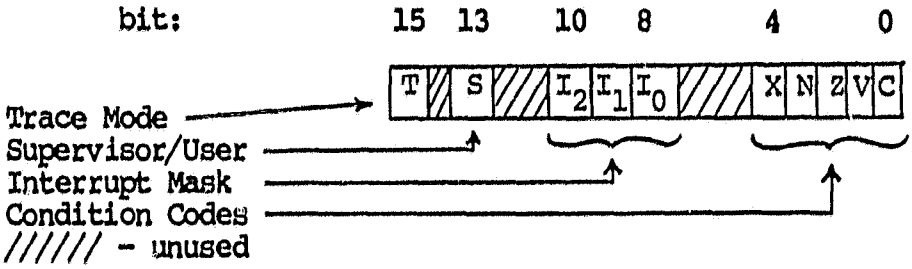
(2) 32-bit Stack Pointers (Supervisor/User)

(1) 16-bit Status Register
Program Counter
16-Mbyte direct addressing range
56 powerful instruction types
Operations on five main data types
Memory mapped I/O
14 Addressing modes

The data registers can be used for byte (8-bits), word (16-bits), or long word (32-bits) data operations. The address registers and stack pointers can be used as base address registers and software stack pointers. These same registers can also be used for word and long word address operations. Any and all of the registers can be used as index registers.

The status register contains the condition codes on the lower byte. The upper byte contains an 8-level priority interrupt masking ability, Supervisor/User state control, and trace mode state control. The trace mode state can be used to single step through a program displaying all registers after each step. There are still six unused bits in the status register. These make room for any extensions of the MC68000 family. Understanding this register is critical in programming.

The condition codes have a direct effect on any testing procedures. It is, therefore, very important to note the effects on the condition codes of any instruction before a decision block in the program. Understanding the interrupt masking system is also important. Whenever an interrupt is serviced, that level of interrupt, and all those below, are automatically masked until a Return from Exemption (RTE) is served. In this way the interrupt remains the higher priority routine. Only an interrupt of higher priority can be serviced at this time. This register is depicted in the following figure:



The 5 main data types mentioned are as follows:

1. Bit
2. BCD Digits (4-bits)
3. Bytes
4. Words
5. Long Words

The 14 addressing modes are derived from the following 6 basic types:

1. Register Direct
2. Register Indirect
3. Absolute
4. Immediate
5. Program Counter Relative
6. Implied

A powerful aspect of the register indirect addressing mode is the capability of post-incrementing, pre-decrementing, offsetting, and indexing. This is by no means a comprehensive look at the MC68000's capabilities. However, it does lend insight into the power and versatility of this one 64-pin chip. Especially useful, and equally complex, is the variety of interrupt and exception processing control available. Combining these with the 1000 plus instruction set leads to quick decision making and data processing.

Some specifics involving the PID's and the PIM can be found in Appendices I and II.

4. DIGITAL LOOP OVERVIEW

The closed loop system as shown in Fig. 2 was modified to accommodate the digital device in the control loop. (See Fig. 6) The motor velocity which is represented by an analog voltage in the analog computer was converted to a

Conceptual System Flow

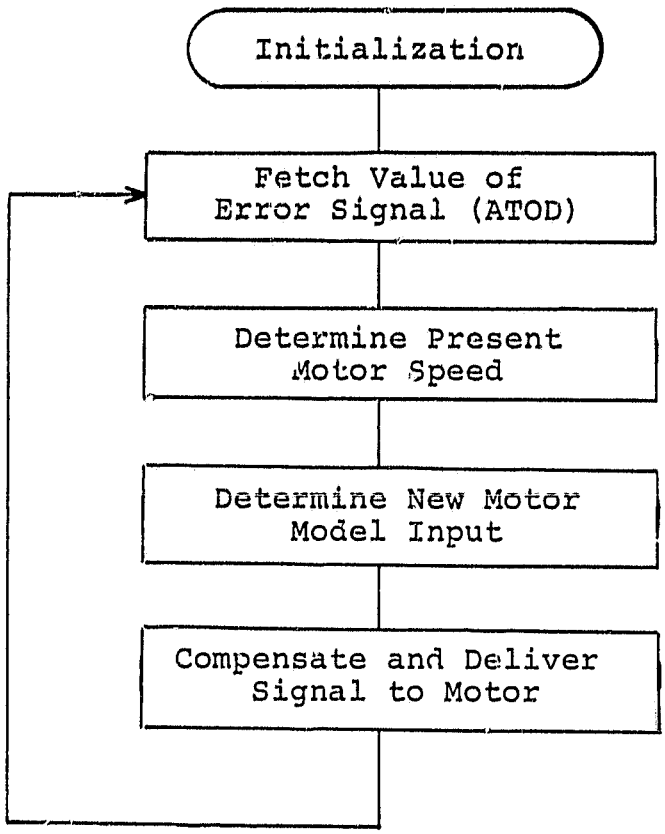


Fig. 6

digital signal by applying it to the input of a voltage-to-frequency converter followed by a D flip flop. The output of the flip flop is a rectangular wave whose pulse length is inversely proportional to the motor velocity. By counting clock pulses gated by this flip flop output, a digital number indicating motor velocity is developed. This process is explained in the sequel (paragraph 6.2).

The motor velocity is compared with a voltage which will be a measure of the desired motor velocity. This voltage is called the reference signal. It will be normalized and digitized as explained (paragraph 6.1) and the comparison made in the microprocessor. This comparison will produce an error signal and this signal will be used to actuate the motor. In the case where a compensating filter is not present, an offset will have to be added to the error which will keep the motor running at a nominal speed when the error is zero. Since the compensating filter adds an integration, the offset can be dropped.

A digital filter was developed to do the compensating. This filter accepts the error signal from the comparison operation and produces the digital equivalent of the actuating voltage. This, after D to A conversion is input to the power amplifier driving the motor.

5. SOFTWARE DESIGN

5.1 Program Initialization

The initialization procedure for the program need only be processed once. This sets up all necessary addresses, ports timers, and any initial values used in the main program. The following description steps through the initialization process. Upon examining the program comments, one can follow the process.

The first part of the initialization involves setting label addresses. It is often easier to remember a label than an address. This is done for all often-used addresses. When programming then only the label need be used. The

assembler will place the correct address. These labels represent control registers, data registers, and scratch-pad memory locations.

The second part of the procedure involves the actual initialization. First, both parts of PID 1 are set as output ports. The control interrupt input lines CA1 and CB1 are set to be positive-edge triggered. After each port is initialized, the peripheral data registers of each is cleared of any "floating data".

PID 2 is initialized in the same manner. However, the most significant bit of port B is set as an input. The remaining seven are set as outputs.

The next step is to program the timers. An explanation of how these are programmed can be found at the end of this report. The final step in this process is to assure the timers are enabled by clearing the least significant bit of timer number 1 (T1).

All that remains is to set initial values for any variables used, and to set memory locations for exception processing routines. In the following program can be found all the initialization for the active programming done. Later in the report routines that were unable to be implemented due to logistic problems will be discussed. Any further initialization needed for these routines will be discussed in their respective sections.

The program is now ready to begin. Once in the main program, this procedure will never be entered again. Instead we will always branch to the head of the main program.

Initialization Program:

LABEL	OP CODE	OPERAND	COMMENTS
PIAD	EQU	$3FF41	SET LABELS
PIAC	"	$3FF45	"
PIBD	"	$3FF43	"
PIBC	"	$3FF47	"
PIBD2	"	$3FF42	"
PIBC2	"	$3FF46	"
PTMCR1	"	$3FF61	"
PTMCR2	"	$3FF63	"
PTMCR3	"	$3FF61	"
PTMC2	"	$3FF69	"
PTMC3	"	$3FF6D	"
MSBB3	"	$3FF6D	"
INIT	"	$6000	"
BUF	"	$6010	"
REF	"	$6020	"
SVBT	"	$6030	"
CLBT	"	$6040	"
	MOVE.B	#%00101010,PIAC	PIAD→ Data dir. register
	MOVE.B	#$FF,PIAD	Config entire port as output
	MOVE.B	#%00101110,PIAC	PIAD→ Data register
	CLR.B	PIAD	
	MOVE.B	#%00101010,PIBC	} Same as PID1-A
	MOVE.B	#$FF,PIBD	
	MOVE.B	#%00101110,PIBC	
	CLR.B	PIBD	
	MOVE.B	#%00101010,PIBC2	
	MOVE.B	#$7F,PIBD2	MSB→ input,remainder out
	MOVE.B	#%00101110,PIBC2	
	CLR.B	PIBD2	
	MOVE.B	#%10010010,PTMCR2	Accesses T3 control register
	MOVE.B	#%10010011,PTMCR3	Sets T3 timing mode
	MOVE.B	#%10010011,PTMCR2	Accesses T1 control register
	MOVE.B	#$ D2,PTMCR1	Enables all timers
	MOVE.W	#$FFFF,INIT	} Sets init value in T3 latch
	MOVE.W	INIT, D0	
	MOVE.L	#MSBB3,A0	
	MOVEP.W	D0,0(A0)	
	MOVE	#$8080,SVBT	Saves a bit in ATOD
	MOVE	#$7F7F,CLBT	Clears a bit in ATOD
	MOVE.L	#AINT,$006C	$006C ←<AINT>
	MOVE.L	#BINT,$0070	$0070 ←<BINT>
	MOVE.L	#BNDCHK,$0018	$0018 ←<BNDCHK>

6. MOTOR SPEED MEASUREMENT

Mounted on the flywheel in the torque system is a printed disk with black and white segments alternately spaced around the circumference. There are exactly 60 black and white pairs evenly spaced. When the flywheel spins an electrooptical pickup produces a rectangular pulse train with the pulse lengths being inversely proportional to the wheel velocity. The voltage to frequency converter coupled with the D flip flop is analogous to this system in the simulation. The value of motor velocity, after this transduction, is to be compared with the reference velocity.

This routine constitutes what might be called the Main Program. For variations of the program additions to or deletions from this routine will be made. Our objective is to sample the reference signal, compare it to the motor speed and feed the difference to the compensator-motor model.

6.1 Reference Service Routine

To determine the signal needed to send back to the motor, use was made of the PTM on board the M68KDM. Of the three timers available, two are used. T2 is used as a continuous clock to regulate the sampling time of the reference signal. The lowest sampling rate while using just one countdown clock is approximately 12.2 Hz. Assuming a maximum sinusoidal input frequency of 100 Hz, this sample time is quite adequate. This sample time corresponds to a latch input of $FFFF when using the internal clock input of 8×10^5 Hz. This clock (T2) will then initiate a priority four interrupt (BINT) every 82 msec. When servicing this interrupt request the subroutine ATOD will be called to take a sample of the reference signal.

ATOD is designed to perform an analog-to-digital conversion on any positive voltage level between 0 and 10. (This routine is discussed in Section 7). Since it is possible to have a reference that is negative, the digital reference

voltage (REF) has to be manipulated in the service routine before it can be used to determine an error in the main program.

A description of this reference signal, and what is actually sampled will help at this point. A graphical explanation is easiest to understand. A typical signal showing maximum and minimum values is in Fig. 7.

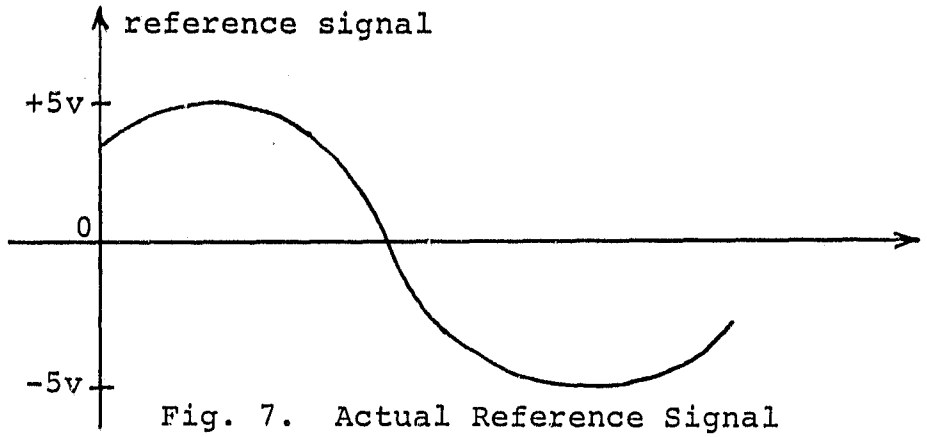


Fig. 7. Actual Reference Signal

Before performing the A/D conversion, we add a +5 volt dc offset to assure a positive signal. (See Fig. 8) The A/D conversion is then made.

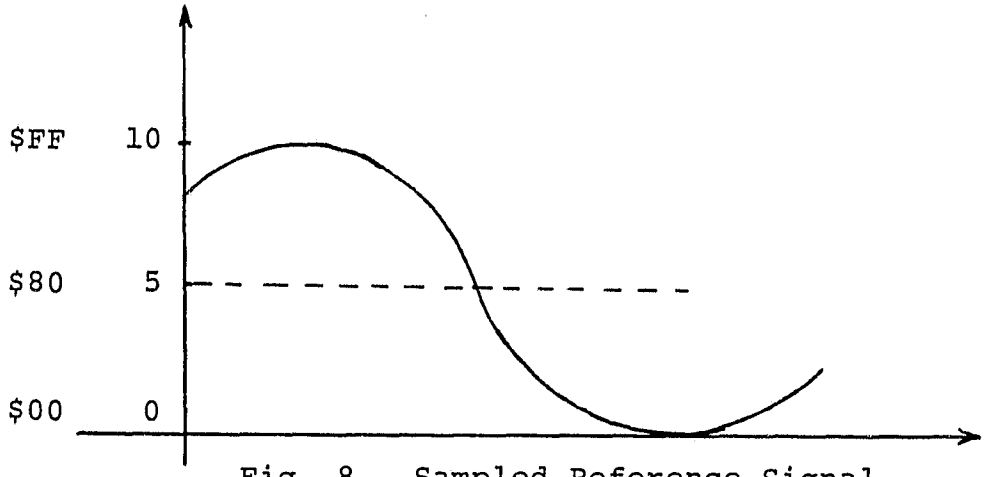


Fig. 8. Sampled Reference Signal

REF from ATOD is a 4-byte word. In order to assure the most significant two bytes are $00 a logic OR operation is performed on REF and $FF00. REF is now a two byte positive digital equivalent to our signal. The manipulation can now be done to give us a workable error.

The zero axis corresponds to $80. Since the conversion takes place on a full two byte scale, it must be scaled down. Otherwise the motor will constantly be over driven. At the same time the offset must be removed to allow a negative error. Hence, the following is done in the service routine.

$$REF \leftarrow \left[\frac{REF - \$80}{\$OA}\right]$$

REF is now a usable digital equivalent of the sampled reference signal. This value will be used in the main program to determine the error to be sent back to the motor.

6.2 Motor Velocity (Main Program)

To determine the motor speed use was made of timer 3 (T3). T3 is used to measure the pulse width of the output of the voltage-to-frequency converter. The length of this pulse is inversely proportional to the speed of the motor. T3 decrements from its initial value ($FFFF) until stopped. This count represents the pulse width. Use was made of T3's option that will enable its clock frequency to be at 10^5 Hz. With the V/F output tied to the gate input of T3 the timer acts as follows. (See Fig. 9)

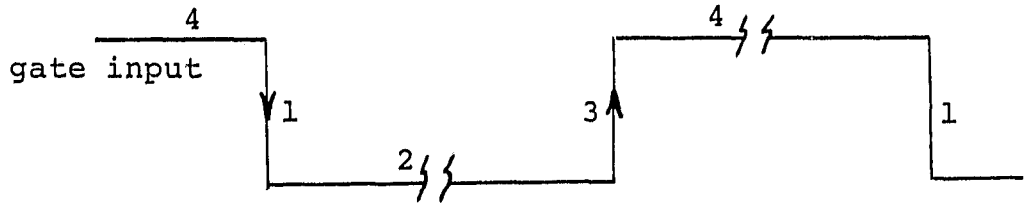


Fig. 9. Velocity Transducer Gating

1. Initial value loaded into T3 ($FFFF).
2. Counter enabled, thus decrementing initial value, 10^5 counts per second.
3. Interrupt triggered, decrementing has stopped.
4. Count is held until reinitialized by (1). Microprocessor fetches count.

The interrupt created by this clock is of priority 3. This assures that a constant sampling time (priority 4) is maintained. The priority 3 interrupt (AINT) increments a flag (DO) and then returns to the main program. In the main program we wait for two of these interrupts (DO=2) before fetching the count from T3. This assures us that we have not begun decrementing before the initial count is set.

Once the proper decremented count is reached, it is subtracted from the initial value, resulting in the actual count. Since this count is inversely proportional to the actual motor velocity, we subtract REF from the count. Before sending this digital voltage back to the motor model, we first must make some changes and check the bounds to be sure the error is between $00-$FF (remember, the D/A converters operate on positive numbers only).

First, $80 must be added to move the zero point to mid-range of the scale. When the compensating integrator is put in the loop, this offset should be removed and the pure error signal fed to the digital filter. After offsetting we use a Check Register Against Bounds command which assures that we do not overflow our D/A range. This command holds the error signal being sent back to the motor to within its respective bounds given an overflow at those bounds. This is a powerful and very useful command. The last step is to send the error signal to the motor model, and then return to the top of the program to repeat the routine.

At this point in the development, testing was interrupted when the development system was called back to NASA-LRC for another task. Before its recall this system was displaying intermittent spurious outputs. Several components were replaced and this problem was corrected.

The testing which was in progress when the equipment was returned would have answered several questions. These concerned the scaling of the ATOD result

and its effect on the motor actuating signal and another method of transduction of the motor velocity.

A flow chart of the main program as well as the exception processing routines is shown in Figs. 10 and 11 respectively. These are followed by the routine itself in assembler code.

Main Program - PULSE WIDTH FLOW

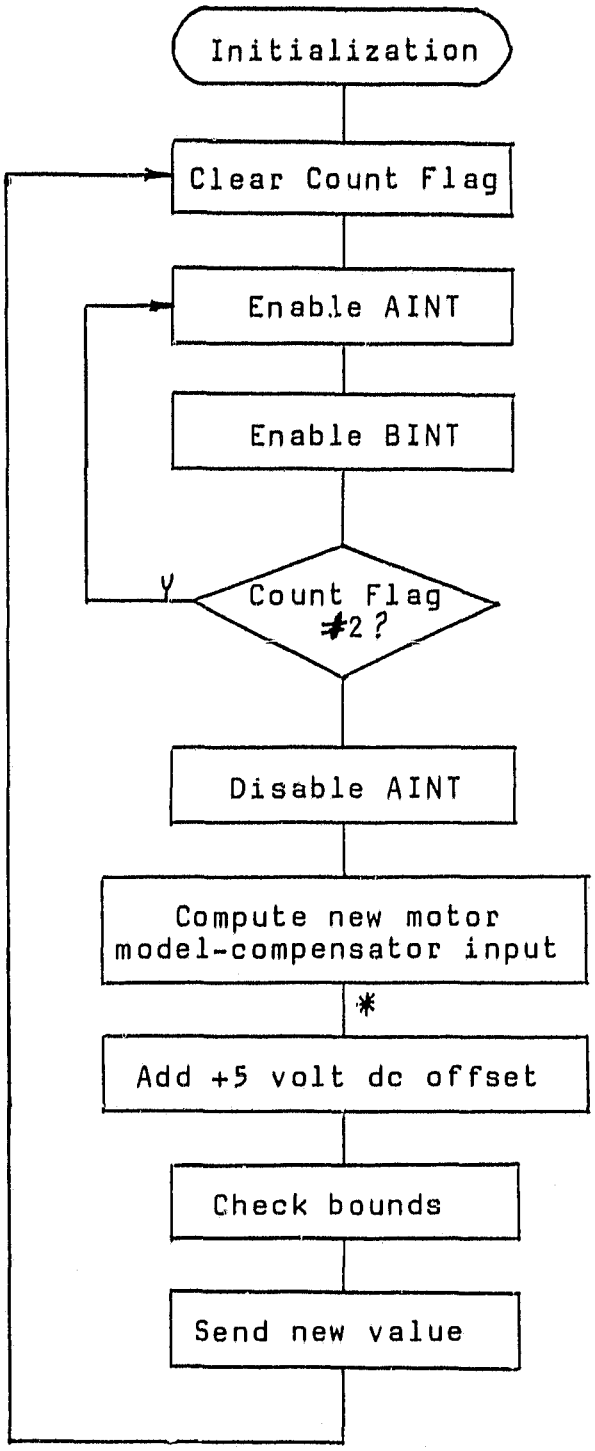
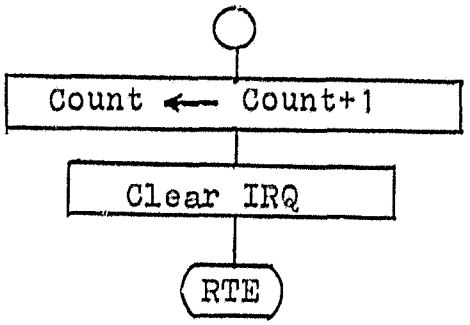
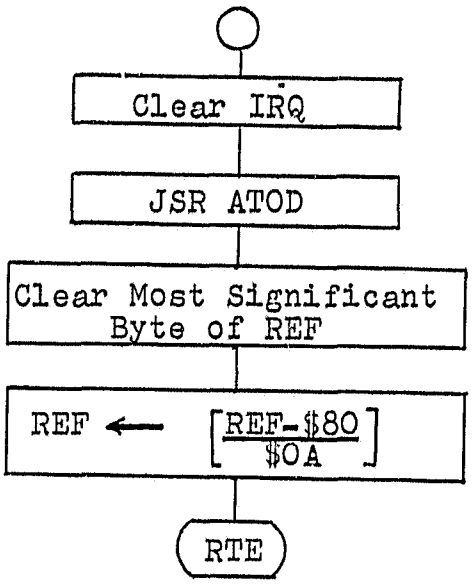


Fig. 10

AINT



BINT



BNDCHK

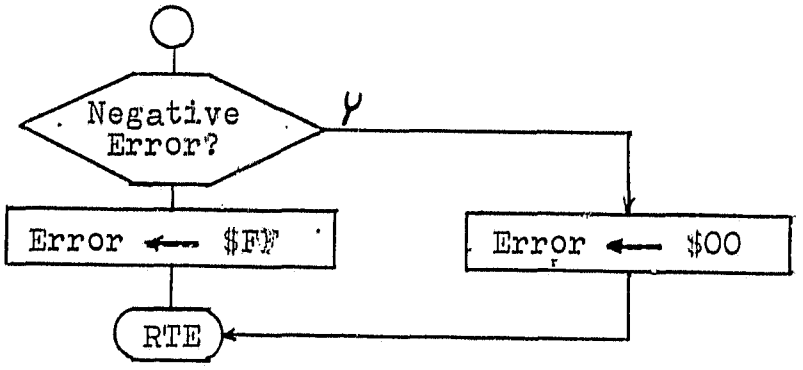


Fig. 11

Main (Pulse Width) Program

LABEL	OPCODE	OPERAND	COMMENTS
START	CLR	D0	Clear count flag
	MOVE.B	#$2F,PIAC	Enable AINT (↓ trigger)
WAIT	MOVE.B	#$2F,PIBC	Enable BINT (↓ trigger)
	CMPI	#2,D0	Count flag = 2?
	BNE	WAIT	If not branch to wait
	MOVE.B	#$2E,PIAC	Mask AINT
	MOVE.L	#PTMC3,A0	} Fetch final value in
	MOVEP.W	0(A0),D2	} T3 to date register D2
	MOVE.W	INIT,BUF	BUF ← INIT
	SUB.W	D2,BUF	BUF ← BUF-D2
	MOVE.W	BUF,D3	D3 ← BUF
	SUB.W	REF,D3	D3 ← D3 - REF
	ADDI	#$0080,D3	D3 ← D3+$80
	CHK	#$00FF,D3	If $FF<D3<$00 JMP to BNDCHK
	MOVE.B	D3,PIBD	Sends error to output port B PID1
	BRA	START	Repeat procedure
AINT	MOVE.B	P1AD,P1AD	Reads PIAD to clear interrupt request
	ADDI	#1,D0	Increment count flag
	RTE		Return from exception
BINT	MOVE.B	PIBD,PIBD	Reads PIBD to clear interrupt request
	ORI	#$FF00,REF	REF ← (REF)OR(FF00)
	MOVE	REF,D7	D7 ← REF
	SUB.T	#$80,D7	} REF ← ⌈(REF-$80)/$0A⌉
	DIVS	#$0A,D7	}
	MOVE.W	D7,REF	}
	RTE		Return from exception
BNDCHK	BMI	NEG	Branch to NEG if error negative
	MOVE.B	#$FF,D3	Otherwise error (D3) ← $FF
	BRA	HOME	Return
NEG	MOVE.B	#$00,D3	Error (D3) ← $00
HOME	RTE		Return from exception

7. ANALOG TO DIGITAL CONVERSION - ATOD

In order to assure a constant conversion time the successive approximation technique for the analog to digital conversion was used. The conversion is based on a 0-10 volt range. This range, using 8 bits, provides the following quantization level:

$$\epsilon = \frac{10}{2^8} = 0.04 \text{ volt}$$

The hardware was available to use 16 bits, however, it can be seen that using only 8 bits is not only sufficient from a quantization level viewpoint, but also saves time. The faster a conversion is made, the more time there is available for other programming.

The method of successive approximation is quite simple. The idea is to successively split the 8 bit digital signal. On each split the signal is sent to a D/A converter where it is compared to the analog signal being sampled. For each test, a decision is made whether to keep the last bit tested. The process is outlined below:

1. The MSB of data is set high.
2. If the comparator output is high, the bit is saved. Otherwise it is cleared.
3. Continue the above process until each successive bit has been tested.

A comparator is used to compare the error signal (after the previously mentioned +5 volt dc offset) and the test signal. The test signal is greater than the error on a high comparator output, and less than the error on a low output. Since the M68KDM operates at such a high speed, a short delay was necessary between sending the test signal and reading the comparator output. This enabled the external circuitry time to perform its function.

Following is a description of the terminology used in both the flow chart and program. The flow chart is shown in Figure 12.

D4 (Data register 4)
D4 contains the output of the comparator. It is read through PID2-8.

D5 (Data register 5)
D5 is used as a counter to determine when all 8-bits have been tested.

D6 (Data register 6)
D6 contains the test signal. Only the least significant 8-bits are used.

SVBT (Save bit)
SVBT is initialized in the initialization routine to $8080. It is used to set up each successive bit to be tested. It must then be rotated to each next bit to be tested. A logical OR operation is then performed on SVBT and the 8-bit test signal.

CLBT (Clear bit)
CLBT is initialized in the initialization routine to $7F7F. It is rotated to follow the bit being tested. When this bit has to be cleared (comparator output low) a logical AND operation is performed on the test signal and CLBT. Though only the least significant 2-bytes are needed, both CLBT and SVBT are 4-bytes to simplify rotation.

REF (Reference voltage)
REF contains the final digital voltage equivalent to the error signal. To simplify programming, REF is constantly updated. Using a number of test and branch statements may speed up the conversion process by addressing REF just once for each sample.

The routine has proven quite accurate and presented few problems. Equipment problems did not permit the opportunity to note the actual conversion time.

ATOD Flow

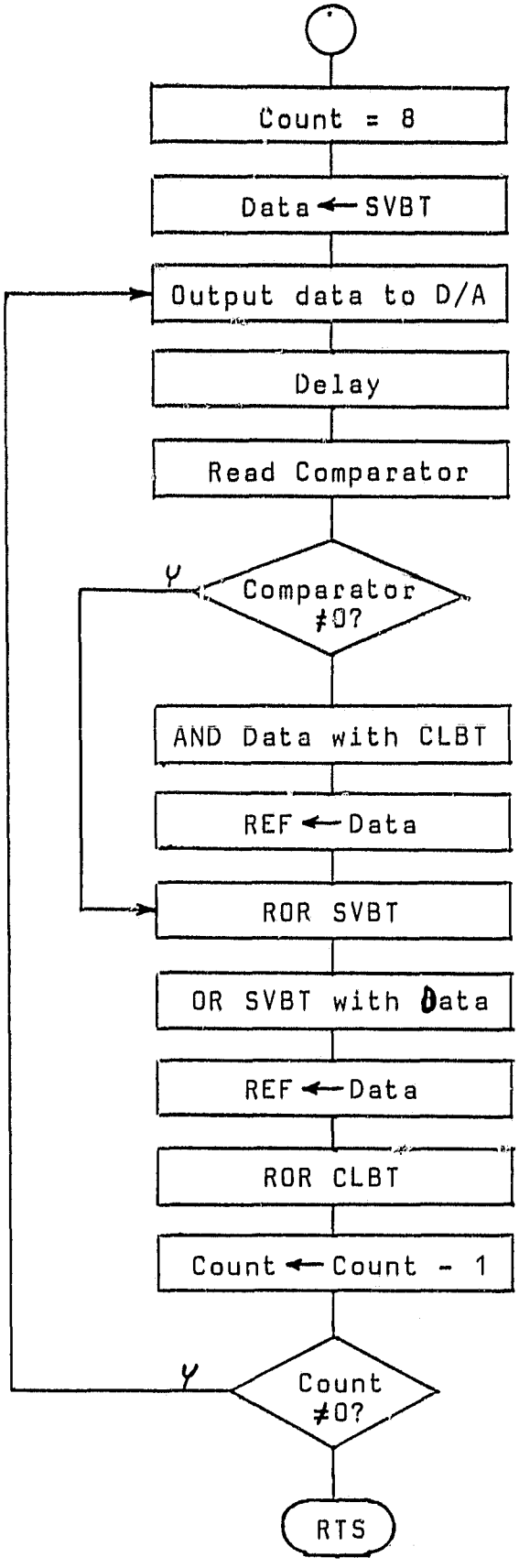


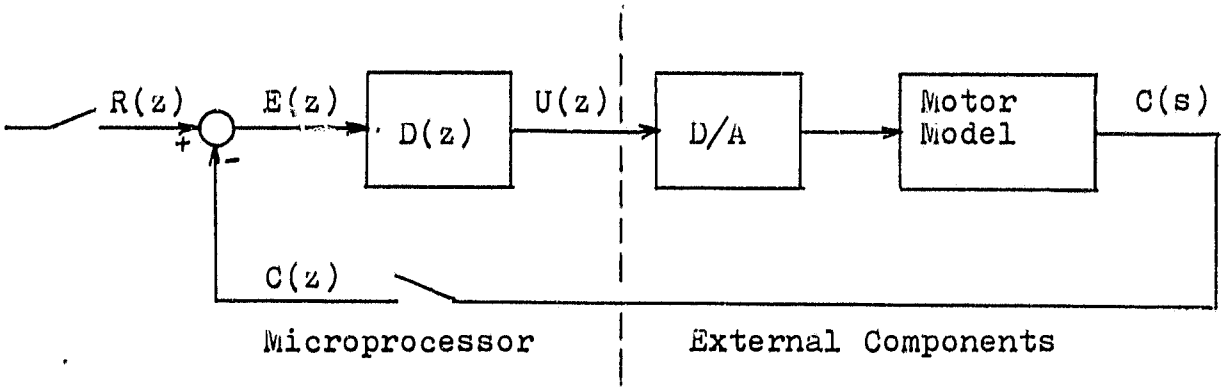
Fig. 12

ATOD Program

```
LABEL   | OPCODE   | OPERAND    | COMMENTS
--------|----------|------------|-----------------------------------
ATOD    | MOVE     |#8,D5       |Set count
        | MOVE     |SVBT,D6     |Set MSB to be tested
LOOP    | MOVE.B   |D6,PIAD     | Send 8-bit test to D/A
        | NOP      |            |
        | NOP      |            |   Delay for external
        | NOP      |            |   circuitry
        | NOP      |            |
        | MOVE.B   |PIBD2,D4    |Read comparator output
        | CMPI     |#0,D4       |Comparator = 0?
        | BNE      |NEXT        |If not save last test bit
        | AND.B    |CLBT,D6     |Clear last test bit
        | MOVE     |D6,REF      |REF ← D6
NEXT    | ROR      |SVBT        |Move to next test bit
        | OR.B     |SVBT,D6     |Set next test bit
        | ROR      |CLBT        |Follow test bit with CLBT
        | SUBI     |#1,D5       |Decrement count
        | CMPI     |#0,D5       |Finished conversion?
        | BNE      |LOOP        |If not continue process
        | RTS      |            |Return from subroutine
```

8. DIGITAL COMPENSATOR

The analog compensator can easily be programmed into the microprocessor without changing the system block diagram significantly. However, more of the functions will be implemented in software as shown below:



Design:

We know $$D(s) = \frac{0.32\ (s+0.04)}{s(s+5)}$$

Using Pole-Zero mapping; $z=e^{sT}$

$$T = 0.082 \text{ sec.}$$

$$D(z) = \frac{K(z-0.9677)(z+1)}{(z-1)(z-0.6637)}$$

Solving for K

$$sD(s)\Big|_{s=0} = (z-1)D(z)\Big|_{z=1}$$

$$\frac{(0.32)(0.04)}{5} = \frac{K(0.0323)(2)}{0.3363}$$

$$K = 0.1333$$

Hence, $$D(z) = \frac{U(z)}{E(z)} = \frac{0.1333(z-0.9677)(z+1)}{(z-1)(z-0.6637)}$$

$$U(z)\left[z^2-1.6637z+0.6637\right] = 0.1333E(z)\left[z^2+0.0323z-0.9677\right]$$

$$U(z)\left[1-1.6637z^{-1}+0.6637z^2\right] = 0.1333E(z)\left[1+0.0323z^{-1}-0.9677z^{-2}\right]$$

This leads to the difference equation:

$$u(k)=0.1333e(k)+0.0043e(k-1)-0.1290e(k-2)+1.6637u(k-1)-0.6637u(k-2)$$

It is immediately apparent that there will be a problem on initially starting our system. The output of this filter, and the

corresponding input to the motor, depends upon previous inputs and outputs. This problem will be handled in the initialization routine. We will set up initial values assuming the motor is operating at its steady state 1000 Hz rate. This then suggests that:

$$\frac{\omega}{V} = \frac{40}{s + .04}$$

at steady state: $s = 0$, $\omega = 1000$

$$\frac{1000}{V} = \frac{40}{.04}$$

$$V = 1$$

At steady state $E(z) = 0$, therefore $e(k) = 0$. Assuming $e(k-2) = u(k-2) = 0$:

$$u(k) = 0.0043\ e(k-1) + 1.6636\ u(k-1)$$

Letting $u(k-1) = e(k-1)$, and $u(k) = 1$

$$u(k-1) = e(k-1) = (0.5995)_{10}$$

The routine is designed to be compatible with the labels and registers used in the existing main program. The digital filter will be inserted just after the error calculation in the main program. This location has been noted on the Pulse Width flow chart with an asterisk.

The multiplication will be done using the ALU board supplied. The documentation for this board is not readily available. Therefore, the multiplication procedure will be displayed as a separate block. The flow chart is shown in Fig. 13 followed by the program in assembler code.

Digital Filter Flow

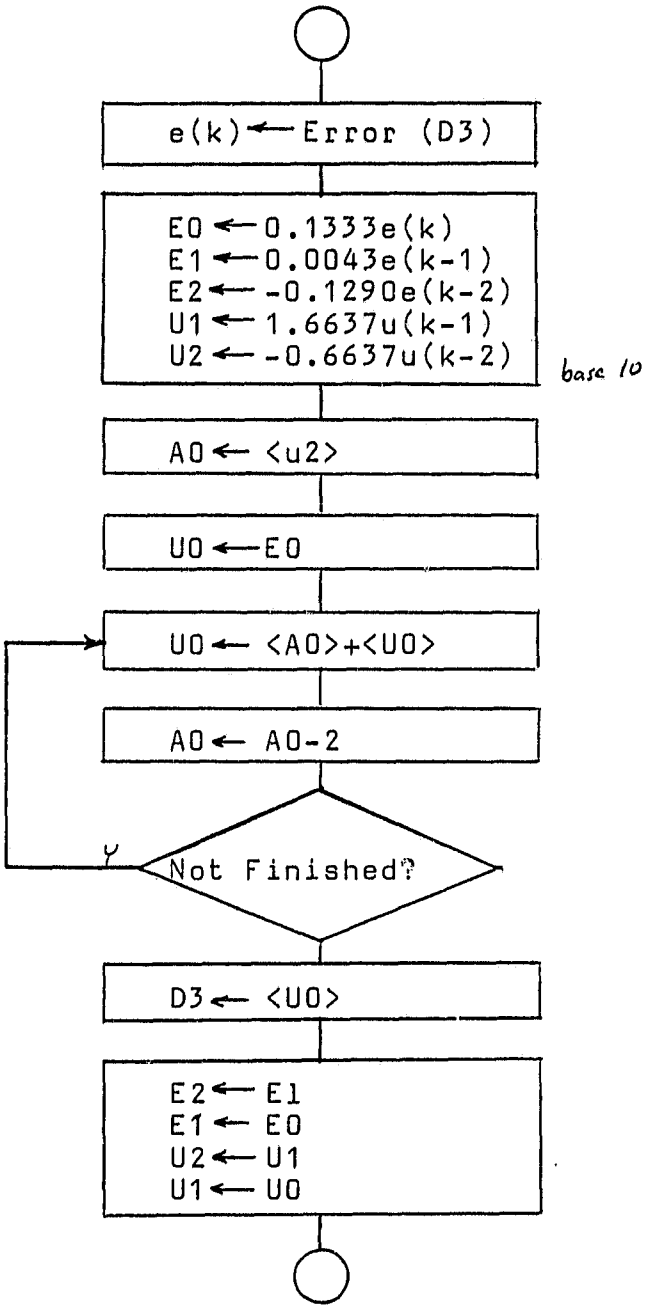


Fig. 13

Compensator Program

```
LABEL   |OPCODE  | OPERAND          | COMMENTS
--------|--------|------------------|-----------------------------------------
E0      |EQU     |$5000             |
E1      |EQU     |$5002             |
E2      |EQU     |$5004             |  This section is in the
U0      |EQU     |$500A             |  initialization routine
U1      |EQU     |$5006             |
U2      |EQU     |$5008             |
        |MOVE.B  |(0.5995)10 ,E1    |
        |MOVE.B  |#$00,E2           |  Initializes filter values
        |MOVE.B  |(0.5995)10 ,U1    |
        |MOVE.B  |#$00,U2           |
        |        |                  |
Filter  |MOVE.B  |D3,E0             |e(k) ← error
        |--------|------------------|
        |E0   (0.1333)10 E0         |
        |E1   (0.0043)10 E1         |
        |E2   (-0.1290)10 E2        |ALU multiplication
        |U1   (1.6637)10 U1         |
        |U2   (-0.6637)10 U2        |
        |--------|------------------|
        |MOVE.B  |U2,A0             |  Sets up first addition
        |MOVE.B  |E0,U0             |
FLOOP   |ADD.B   |A0,U0             |Add
        |SUB.I   |#2,$A0            |Next addition
        |CMPI    |#$5000,$A0        |Finished?
        |BNE     |FLOOP             |If not branch to continue
        |MOVE.B  |U0,D3             |Data register 3 gets filter output
        |MOVE.B  |E1,E2             |
        |MOVE.B  |E0,E1             |  Set values for next sample time
        |MOVE.B  |U1,U2             |  difference equation calculation
        |MOVE.B  |U0,U1             |
```

9. COMMENTS ON THE SYSTEM

The procedures and programming discussed assume the motor is running near its steady state speed. A separate start-up routine will have to be devised to accomplish this.

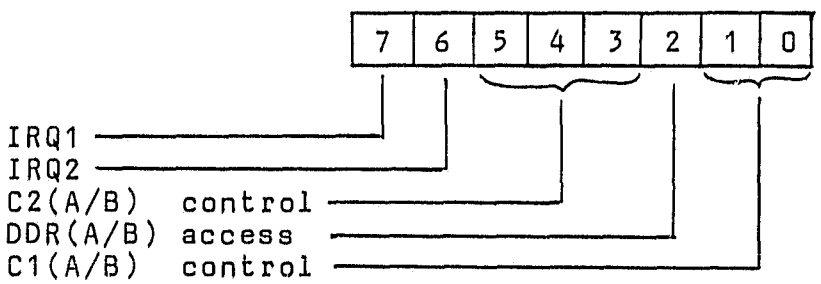
There were some problems encountered on the project. Exception processing initially proved to be quite a challenge. The documentation supplied did not clearly explain how the MPU determines a vector address for servicing these exceptions. Since most of our control depends on interrupt servicing, the problem had to be overcome before any real progress was made. With help from the people at NASA-LRC, the problem was solved. Understanding the Trap Error exception was even more difficult than that of the interrupt exception. For most of the different Trap Errors possible the vector address for the service is floating. It does not remain the same, but changes with respect to the program counter and previous instruction handled.

Wirewrapping on the M68KDM was not a big problem. The documentation was vague here also, but examination of the board's schematic diagram answered these questions. The IRQ lines and inputs had to be wirewrapped to the peripheral devices and the MPU. The data sheets on these peripheral devices provided clear and concise instructions for their use and programming. There were only a few small problems involving the instruction set. Some of the peripheral commands, subtraction, and division required careful manipulation. Except for the material on exception processing the system documentation was generally satisfactory.

Appendix I

Peripheral Interface Device (PID)

There are six addressable registers within each PID, three for each of the two ports. Ports A and B each contain a data direction register (DDR), control register (CR), and a peripheral data register (PDR). Each port also has an interrupt input control line (CA1/CB1) and an input/output peripheral control line (CA2/CB2). The DDR and PDR each share the same memory address. Bit-2 of the control register determines which of these is being addressed. This control register is programmed to determine how the control lines are used.
Control Register (A/B)



Bit-0	1	Interrupt enabled
	0	Interrupt masked
Bit-1	1	Interrupt positive edge triggered
	0	Interrupt negative edge triggered
Bit-2	1	DDR selected
	0	PDR selected
Bits-3,4,5	Determine control and interrupt reaction to CA2/CB2. This control differs slightly for each port.	
Bits-6,7	Set on active transition of respective interrupt and control signal. These interrupt requests can only be reset by a read peripheral data operation on that port.	

To initialize each port, the following procedure is used:

1. Select DDR
2. Program input/output lines
3. Program control lines, select PDR
4. Set initial data values at port

Appendix II

Programmable Timer Module (PTM)

The PTM contains three separate timers. The three operate independently of each other and are essentially the same. Each can use the internal 0.8 MHz clock source or an external clock. Each has accessible:

(1) 8-bit Control Register
(1) 16-bit Latch
(1) 16-bit Decrementing Counter

T3 has the added option of a divide clock by 8 function. This was used along with the internal clock to produce a 0.1 MHz clock. The latch is used to preset the count for the timer. The counter, when enabled, is then decremented from this initial value until disabled. The gate input is used to trigger the counter.

There are a number of control word formats which set up different timing mechanisms. The PTM is very useful, in that it operates independently of the microprocessor, thus, freeing the microprocessor for other tasks. Upon comleting its function, the timer signals the MPU to fetch its data, and may then continue another function.

The control registers for T1 and T3 share the same address. T2's control register selects which is being addressed. An explanation of the register control notation follows:

CR1	T1	Control Register
CR2	T2	Control Register
CR3	T3	Control Register
CRX	All	Three Control Registers

The number immediately following these three characters refers to the specific bit in the control register. The following figure depicts the individual bit assignments for any given timer function.

TIMER PROGRAMMING

BIT	VALUE	DESCRIPTION
CR10	0	All timers enabled
	1	Timers held at preset state
CR20	0	CR2 may be written to
	1	CR1 may be written to
CR30	0	T3 clock not prescaled
	1	T3 clock prescaled by ÷ 8
CRX1		Clock Source
	0	External clock
	1	Internal clock
CRX2		Counting Mode Control
	0	Normal 16-bit mode
	1	Dual 8-bit mode
CRX3,4,5		Counter Mode and Interrupt Control (see below)
CRX6		Interrupt Enable
	0	Interrupt masked
	1	Interrupt enabled
CRX7		Output Enable
	0	Output masked
	1	Output enabled

CRX3	CRX4	CRX5	Timer Operating Mode
0	*	0	Continuous
0	*	1	Single Shot
1	0	*	Frequency Comparison
1	1	*	Pulse Width Comparison

(* determines additional timer functions)

Appendix III

SYSTEM MEMORY MAP